VSC faults: generalization to multiple converters

CITCEA

Josep Fanals

04/2021

1 Development

We present a fundamental scheme in Figure 1, from where we will extend the system. The goal here is to show a scalable procedure to solve the circuit for all possible faults. The analysis is performed in the *abc* natural frame of reference, where the constraints are set. Then, voltages are expressed in symmetrical components and the optimization function is called. The grid equivalent is depicted as a Norton equivalent for greater simplicity in the analysis.

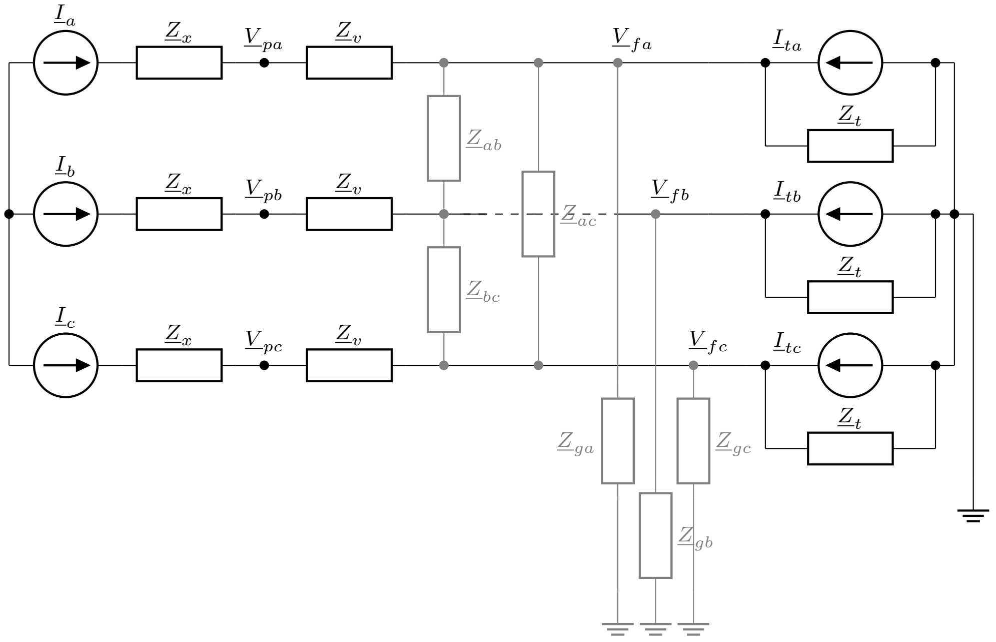

FIGURE 1. General *abc* scheme to analyze faults

In matricial form, the equations that define the system depicted in 1 become:

$$
\begin{pmatrix} \underline{I}_a \\ \underline{I}_b \\ \underline{I}_c \\ \hline \underline{I}_{ta} \\ \underline{I}_{tb} \\ \underline{I}_{tc} \end{pmatrix}
=
\left(\begin{array}{ccc:ccc}
\underline{Y}_v & 0 & 0 & -\underline{Y}_v & 0 & 0 \\
0 & \underline{Y}_v & 0 & 0 & -\underline{Y}_v & 0 \\
0 & 0 & \underline{Y}_v & 0 & 0 & -\underline{Y}_v \\
\hdashline
-\underline{Y}_v & 0 & 0 & \underline{Y}_{f,11} & \underline{Y}_{f,12} & \underline{Y}_{f,13} \\
0 & -\underline{Y}_v & 0 & \underline{Y}_{f,21} & \underline{Y}_{f,22} & \underline{Y}_{f,23} \\
0 & 0 & -\underline{Y}_v & \underline{Y}_{f,31} & \underline{Y}_{f,32} & \underline{Y}_{f,33}
\end{array}\right)
\begin{pmatrix} \underline{V}_{pa} \\ \underline{V}_{pb} \\ \underline{V}_{pc} \\ \hline \underline{V}_{fa} \\ \underline{V}_{fb} \\ \underline{V}_{fc} \end{pmatrix},
\tag{1}
$$

where the admitances, denoted by $\underline{Y}$, are the inverse of the corresponding impedances $\underline{Z}$. The elements of the form $\underline{Y}_{f,ij}$ form the matrix $\underline{\mathbf{Y}}_{\mathbf{f}}$, which turns out to be:

$$\underline{\mathbf{Y}}_{\mathbf{f}} = \begin{pmatrix} \underline{Y}_{ab} + \underline{Y}_{ac} + \underline{Y}_{ga} + \underline{Y}_{t} + \underline{Y}_{v} & -\underline{Y}_{ab} & -\underline{Y}_{ac} \\ -\underline{Y}_{ba} & \underline{Y}_{ba} + \underline{Y}_{bc} + \underline{Y}_{gb} + \underline{Y}_{t} + \underline{Y}_{v} & -\underline{Y}_{bc} \\ -\underline{Y}_{ca} & -\underline{Y}_{cb} & \underline{Y}_{ca} + \underline{Y}_{cb} + \underline{Y}_{gc} + \underline{Y}_{t} + \underline{Y}_{v} \end{pmatrix}. \tag{2}$$

With this approach, the solution to the system is rather straightforward. Solving for the vector formed by $\underline{V}_p$ and $\underline{V}_f$ involves moving to the left hand side of the equation the second summand found in Equation 1, calculating the inverse of the admittances matrix that multiplies the unknowns and computing the final product.

The advantage of attacking the problem this way, and not in the previous manner where we developed the particular expressions for each fault, is found in its flexibility. Each type of fault can be simulated, and even, simultaneous faults can be calculated. It also makes sense to develop the problem in this way due to the fact that including another converter does not add much complexity. We are now going to add another converter, which we will call VSC2; VSC1 is supposed to be the already present converter shown in Figure 1. This new converter remains connected to the point where the fault takes place. Our goal is to obtain the necessary equations to compute the particular voltages at the PCC of each branch together with the voltage at the faulted point. Just for clarification purposes, Figure 2 shows the single-phase representation of the system with the two converters added to the faulted grid.

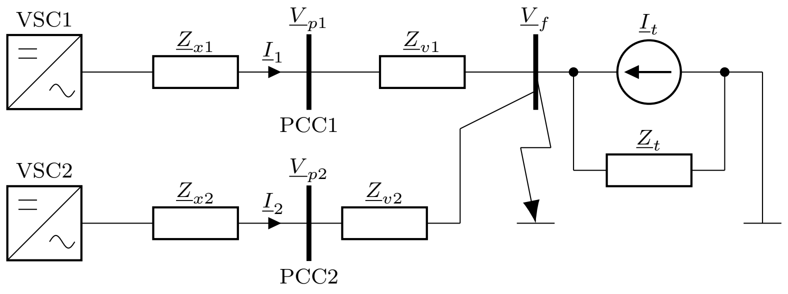

FIGURE 2. Single-phase representation of the simple system under a fault

Similarly to the single converter scenario, the Kirchoff equations in compact form are:

$$\begin{pmatrix} \underline{\mathbf{I}}_{\mathbf{1}} \\ \underline{\mathbf{I}}_{\mathbf{2}} \\ \underline{\mathbf{I}}_{\mathbf{t}} \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{Y}}_{\mathbf{v1}} & \mathbf{0} & -\underline{\mathbf{Y}}_{\mathbf{v1}} \\ \mathbf{0} & \underline{\mathbf{Y}}_{\mathbf{v2}} & -\underline{\mathbf{Y}}_{\mathbf{v2}} \\ -\underline{\mathbf{Y}}_{\mathbf{v1}} & -\underline{\mathbf{Y}}_{\mathbf{v2}} & \underline{\mathbf{Y}}_{\mathbf{f}} \end{pmatrix} \begin{pmatrix} \underline{\mathbf{V}}_{\mathbf{p1}} \\ \underline{\mathbf{V}}_{\mathbf{p2}} \\ \underline{\mathbf{V}}_{\mathbf{f}} \end{pmatrix}, \tag{3}$$

where the vectors $\underline{\mathbf{I}}_{\mathbf{1}}$ and $\underline{\mathbf{I}}_{\mathbf{2}}$ denote the *abc* currens injected by the associated converter; $\underline{\mathbf{Y}}_{\mathbf{v1}}$ and $\underline{\mathbf{Y}}_{\mathbf{v2}}$ are matrices full of zeros expect for the elements $\underline{Y}_{v1}$ and $\underline{Y}_{v2}$ placed on the diagonal; $\underline{\mathbf{Y}}_{\mathbf{f}}$ is the matrix shown in Equation 2, where $\underline{Y}_v$ would be the sum of $\underline{Y}_{v1}$ and $\underline{Y}_{v2}$ in this case; $\underline{\mathbf{V}}_{\mathbf{p1}}$ and $\underline{\mathbf{V}}_{\mathbf{p2}}$ are the *abc* voltages at the point of common coupling of each branch; $\underline{\mathbf{Y}}_{\mathbf{t}}$ is the diagonal matrix with $\underline{Y}_t$ terms; and $\underline{\mathbf{V}}_{\mathbf{t}}$ is the voltage imposed by the grid, in the *abc* frame.

Adding more converters to the grid would not suppose a challenge from the point of view of solving the system since the general structure of the equations would take the same form. A more general case could consist of adding a grid between the point where the converters are connected and the Thevenin equivalent. The approach we suggest here would be the same as the one shown in Equation 3, where the complete matrices are formed by concatenating 3×3 block matrices. At this stage, the block matrices are formed by inspection, but the positive aspect of employing admittance matrices is that their construction can be easily programmed. Thus, we believe that the procedure shown here is meant to be adaptable to an n converter case with a full grid, as Figure 3 tries to illustrate. For now, we will suppose the grid to be fully passive, in the sense that there are no power loads or generator

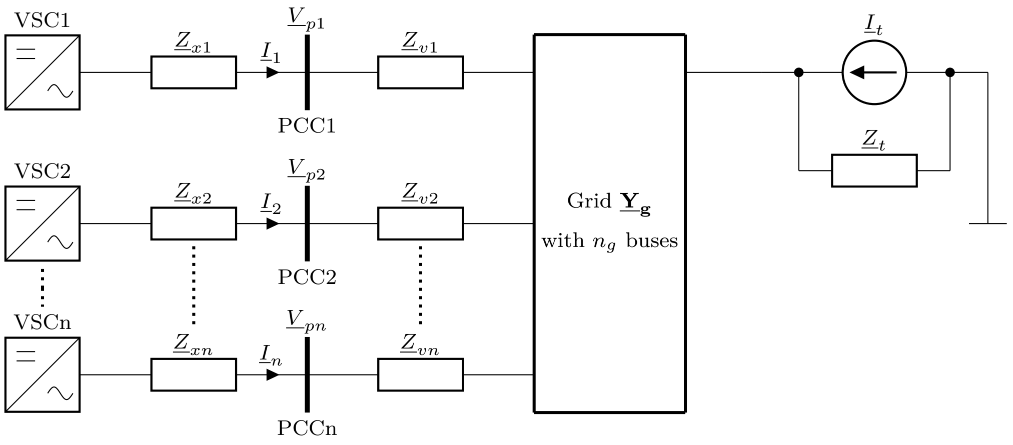

Figure 3. Single-phase representation of a complete system

units connected to it. Consequently, it can be described by only an admittance matrix, which causes the development of the equations to be considerably simpler.

As follows, the equations to solve the system in matricial form are presented below:

$$\begin{pmatrix} \underline{\mathbf{I}}_{\mathbf{1}} \\ \underline{\mathbf{I}}_{\mathbf{2}} \\ \vdots \\ \underline{\mathbf{I}}_{\mathbf{n}} \\ \underline{\mathbf{I}}_{\mathbf{t}} \end{pmatrix} = \begin{pmatrix} \underline{\mathbf{Y}}_{\mathbf{v1}} & \mathbf{0} & \cdots & \mathbf{0} & -\underline{\mathbf{Y}'}_{\mathbf{v1}} \\ \mathbf{0} & \underline{\mathbf{Y}}_{\mathbf{v2}} & \cdots & \mathbf{0} & -\underline{\mathbf{Y}'}_{\mathbf{v2}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \underline{\mathbf{Y}}_{\mathbf{vn}} & -\underline{\mathbf{Y}'}_{\mathbf{vn}} \\ -\underline{\mathbf{Y}'}_{\mathbf{v1}} & -\underline{\mathbf{Y}'}_{\mathbf{v2}} & \cdots & -\underline{\mathbf{Y}'}_{\mathbf{vn}} & \underline{\mathbf{Y}}_{\mathbf{g}} \end{pmatrix} \begin{pmatrix} \underline{\mathbf{V}}_{\mathbf{p1}} \\ \underline{\mathbf{V}}_{\mathbf{p2}} \\ \vdots \\ \underline{\mathbf{V}}_{\mathbf{pn}} \\ \underline{\mathbf{V}}_{\mathbf{g}} \end{pmatrix}, \tag{4}$$

where matrices of the form $\underline{\mathbf{Y}'}_{\mathbf{x}}$ differ from $\underline{\mathbf{Y}'}_{\mathbf{x}}$ in the sense that they are not 3×3 matrices but instead are constituted by null elements except for the ones that symbolize the connection between a generic bus with the associated $\underline{V}_{px}$ voltage and the grid. Besides, the matrix $\underline{\mathbf{Y}}_{\mathbf{g}}$ includes the $\underline{Y}_t$ admittance and the admittances of the form $\underline{Y}_{vk}$.

2 Two converter case study

Having presented the general formulation, we now describe the optimization problem exemplified for two converters. We focus on the expressions involved, but more than that, on the programming, as it has been tested to be the critical step. In fact, the tools that worked for a single converter scenario have not always been suitable for two converters.

For a system following the notation depicted in Figure 2, the initial optimization problem would read as

$$\begin{aligned} \min_{\underline{I}_1,\underline{I}_2} \quad & \lambda_1^+(1-|\underline{V}_{p1}^+|)^2+\lambda_1^-(0-|\underline{V}_{p1}^-|)^2+\lambda_2^+(1-|\underline{V}_{p2}^+|)^2+\lambda_2^-(0-|\underline{V}_{p2}^-|)^2 \ , \\ \text{s.t.} \quad & \max(\underline{I}_{a1},\underline{I}_{b1},\underline{I}_{c1},\underline{I}_{a2},\underline{I}_{b2},\underline{I}_{c2}) \leq I_{\max} \ , \\ & \sum \underline{I}_1 = 0 \ , \\ & \sum \underline{I}_2 = 0 \ , \end{aligned} \tag{5}$$

where to simplify the notation it has been obviated that the voltages are functions of the currents $\underline{I}_1$ and $\underline{I}_2$. The λ factors serve the purpose to weight the various positive and negative voltage to establish the desired priority. In practice are expected to be set between 0 and 1. We also impose that none of the phase currents can surpass a given maximum current. Such value could be different for each converter. However, we will proceed assuming the two converters have identic current capabilities. In addition to all this, since the VSC can only inject positive and negative sequence, we have to enforce that the summation of currents for each converter becomes null. Otherwise the optimization solver may offer an unfeasible solution.

Notice that absolute values have been employed in Equation 5. It may be preferable to avoid such operator because it is non-linear. Despite the inconvenience, as far as I can tell there exists no linear alternative. Squaring the voltages is not an option because they are complex magnitudes and hence some imaginary part would still be left. Furthermore, computing the absolute value with the complex conjugate would imply the replacement of a non-linear operator with another non-linear operator, so it does not seem to be advantageous. We have tested the response with both and came to the conclusion that opting for the complex conjugate results in slightly shorter computation times. As a result, the optimization problem is rewritten as:

$$\begin{aligned} \min_{\underline{I}_1,\underline{I}_2} \quad & \lambda_1^+(1-\underline{V}_{p1}^+\underline{V}_{p1}^{+*})^2+\lambda_1^-(0-\underline{V}_{p1}^-\underline{V}_{p1}^{-*})^2+\lambda_2^+(1-\underline{V}_{p2}^+\underline{V}_{p2}^{+*})^2+\lambda_2^-(0-\underline{V}_{p2}^-\underline{V}_{p2}^{-*})^2 \ , \\ \text{s.t.} \quad & \max(\underline{I}_{a1},\underline{I}_{b1},\underline{I}_{c1},\underline{I}_{a2},\underline{I}_{b2},\underline{I}_{c2}) \leq I_{\max} \ , \\ & \sum \underline{I}_1 = 0 \ , \\ & \sum \underline{I}_2 = 0 \ . \end{aligned} \tag{6}$$

There are many possibilities to solve the optimization problem. In the beginning we opted for the `scipy.optimize` package, which contains the `minimize()` function. Initially we chose the SLSQP solver. This is aimed at minimizing a scalar function with one or more variables by means of the Sequential Least Squares Programming approach [**2020SciPy-NMeth**]. Such method worked fine for one single converter but struggled to provide a successful optimization with two converters. It had a hard time converging towards the same trend, and thus, it kept jumping around different objective function values for tiny variations in the input parameters. Not only this, but also the theoretically optimal currents to be injected by the VSC presented heavy variations. They could be regarded as noise with the associated problem of leading to erroneous conclusions.

Consequently, we looked and eventually found an alternative based on Python as well: the `mystic` framework. According to the authors, `mystic` is a highly-constrained non-convex optimization and uncertainty quantification tool [**mckerns2012building**, **mckernsmystic**]. Even though we are not interested in the uncertainty quantification by now, the fact that the package is able to solve hard optimization problems becomes the key point. We have reasons to believe that if the `scipy.optimize` package had trouble optimizing a case with two converters, it would most likely fail in solving a system with n converters. The final aim of this work is precisely to solve a case with multiple converters, so relying on a robust optimization package becomes a necessity.

Nevertheless, the optimization formulation is less straightforward as before because in order to solve the problem in a reasonable time period, the constraints have been divided in two parts:

- Hard constraints: they impose $\sum \underline{I}_1 = 0$ and $\sum \underline{I}_2 = 0$. It is mandatory to ensure no homopolar current is present because otherwise the model would not be a realistic representation of the actual VSC.

- Soft constraints: they impose $\max(\underline{I}_{a1}, \underline{I}_{b1}, \underline{I}_{c1}, \underline{I}_{a2}, \underline{I}_{b2}, \underline{I}_{c2}) \leq I_{\max}$, but in contrast with the hard constraints, initially we allow them be violated.

Soft constraints are usually called penalty functions because violating them comes with a price. In this particular case, surpassing the $I_{\max}$ current would mean that the objective function would be severely penalized. As a result, the solver would discard such solutions and look for another path.

We think that using a penalty function for restricting the currents makes complete sense. In reality, injecting a current of 1000.1 A instead of a hypothetical maximum of 1000.0 A would not cause devastating effects, as there is likely to be a safe margin of operation. Uncertainty in measurement equipment may provoke larger errors. Thus, it seems to us that real current limitations resemble soft constraints rather than hard constraints. In addition to that, we have made sure that the program respects the limitations once the optimization calculation is finished. When looked at it from this perspective, soft constraints are a convenient tool to facilitate the obtention of the optimal solution.

With the goal of clarifying the modifications to the code, we show the most representatives added lines of code. First, the soft constraints:

```
def penalty_mean(x, target):
        a1 = x[0] ** 2 + x[1] ** 2
        a2 = x[2] ** 2 + x[3] ** 2
        a3 = x[4] ** 2 + x[5] ** 2
        a4 = x[6] ** 2 + x[7] ** 2
        a5 = x[8] ** 2 + x[9] ** 2
        a6 = x[10] ** 2 + x[11] ** 2
        return (max(a1, a2, a3, a4, a5, a6) - target)

    @quadratic_inequality(condition=penalty_mean, kwds={'target':1.0})
    def penalty(x):
        return 0.0
```

Listing 1. Penalty function definition to softly respect the current limitations

The variables from `a1` to `a6` represent the squared absolute values of the currents. The same could be expressed in a vectorized function, but for now we have preferred to ease the reading. In any case, the penalty function `penalty(x)` returns a value of zero since the constraint is implicitly programmed as `max(a1, a2, a3, a4, a5, a6) - target <= 0`. Hence, when the maximum phase current meets the target of 1 pu (this value is what corresponds to $I_{\max}$), we return a null value. On top of the penalty function there is a decorator, which becomes essentially to extend the functionality of the penalty function so as to treat it as an inequality constraint.

On the other hand, the hard constraints are written in Python as:

```
equations_c = """
    x0 + x2 + x4 == 0
    x1 + x3 + x5 == 0
    x6 + x8 + x10 == 0
    x7 + x9 + x11 == 0
    """

    cf = generate_constraint(generate_solvers(simplify(equations_c))
```

Listing 2. Hard constraints to impose a null homopolar current

There are in fact four constraints because the null summation of currents has to be respected for both real and imaginary part in the two converters.

Finally, we call the optimization solver with:

```
    result = fmin(obj_fun, x0=Ii_t, bounds=bnds, penalty=penalty, constraints=cf, npop=10000, gtol
     =10000, disp=True, full_output=True, ftol=1e-14, maxiter=15000, maxfun=15000)
```

Listing 3. Line to call the optimization

The most relevant parameters required by the `fmin()` function are shortly described below:

- `obj_fun` is the objective function as shown in Equation 6.
- `x0` stands for the initialization array. We have passed to it the previous optimal point.
- `bounds` represent the boundaries of the currents, that is, the limits that enclose the range of allowed values for the currents to optimize.
- `penalty` and `constraints` take the penalty and the constraint functions defined before.
- `npop` is the size of the test solution population while `gtol` defines the maximum number of iterations without improvement.
- `ftol` becomes the maximum allowed relative error to ensure convergence.
- `maxiter` and `maxfun` represent respectively the maximum number of iterations and evaluations of the objective function.

With the main aspects relative to the programming covered, we proceed to show the results for several faults. In all of them we vary the fault impedance and display the currents, the voltages, and the objective function evolution. Unless noted otherwise, the initial parameters are tabulated in Table 1.

Magnitude	**Value**
$\lvert\underline{V}_t\rvert$	1.00
$I_{\max}$	1.00
$\underline{Z}_{v1}$	$0.01 + j0.05$
$\underline{Z}_{v2}$	$0.02 + j0.06$
$\underline{Z}_t$	$0.01 + j0.10$
$[\lambda_1^+, \lambda_1^-, \lambda_2^+, \lambda_2^-]$	[1.00, 1.00, 1.00, 1.00]

Table 1. Initial parameters to study the faults, all in p.u.

3 Plots

3.1 Balanced fault

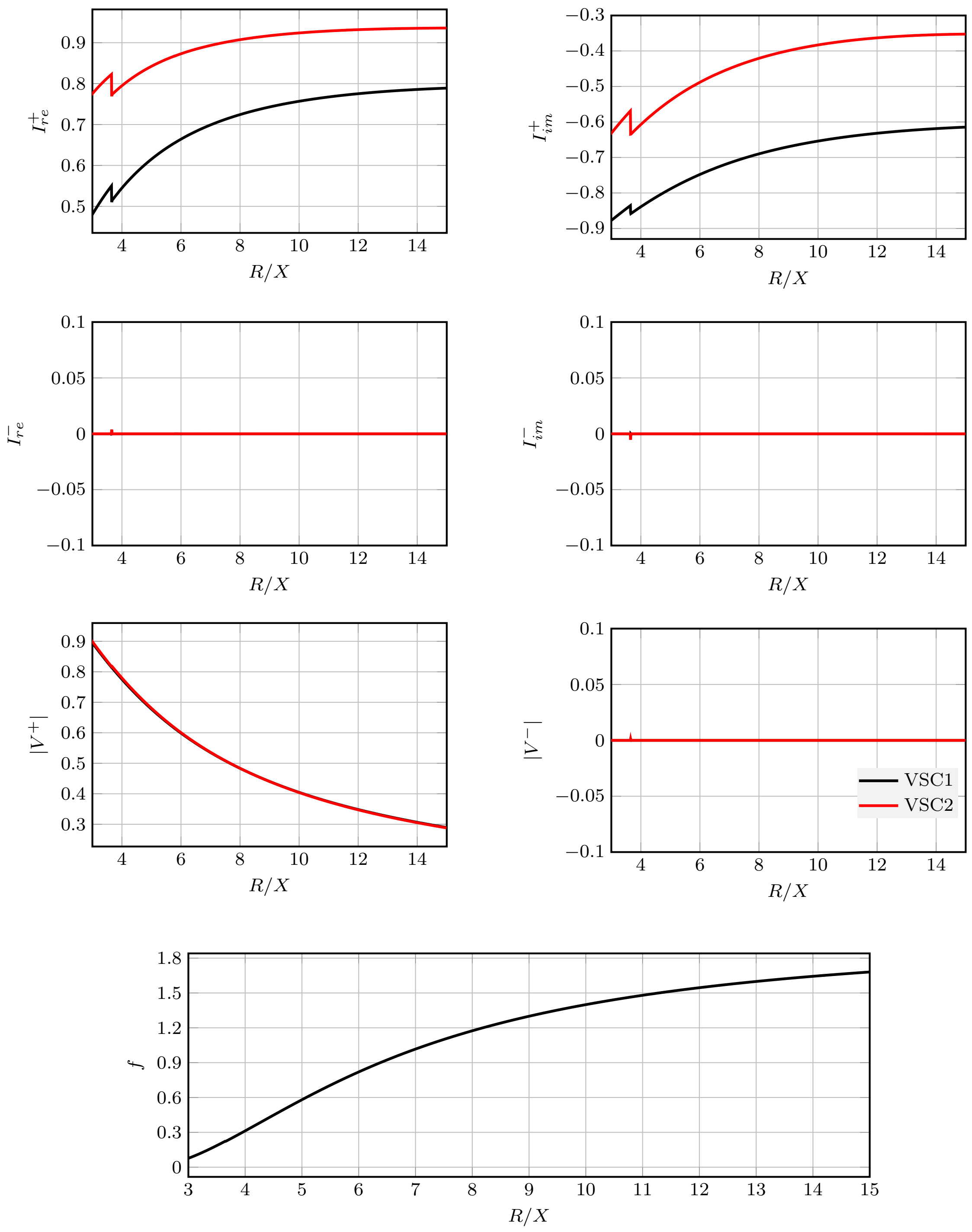

FIGURE 4. Influence of the currents on the objective function for the balanced fault with $\underline{Y}_{xg} = [3, 15]$